Lessons Learned

Solving script organization, one script at a time

Problem Statement

As a software development enthusiast I find myself spontaneously writing scripts to solve immediate problems, which then get dumped in some obscure folder. Over time these scripts can be difficult to manage efficiently, I frequently find them scattered amongst various folders and lost in the clutter of poor management. Several years from now the quantity of scripts will grow such that they will be completely lost in the shuffle and their utility all but vanished. The Lessons Learned database system will offer a solution to this ongoing struggle and transform an unmanageable conglomerate of files into a solid cohesive structure that will improve their usefulness and ensure their longevity. Additionally, the 'final' product will serve as a useful portfolio while seeking employment.

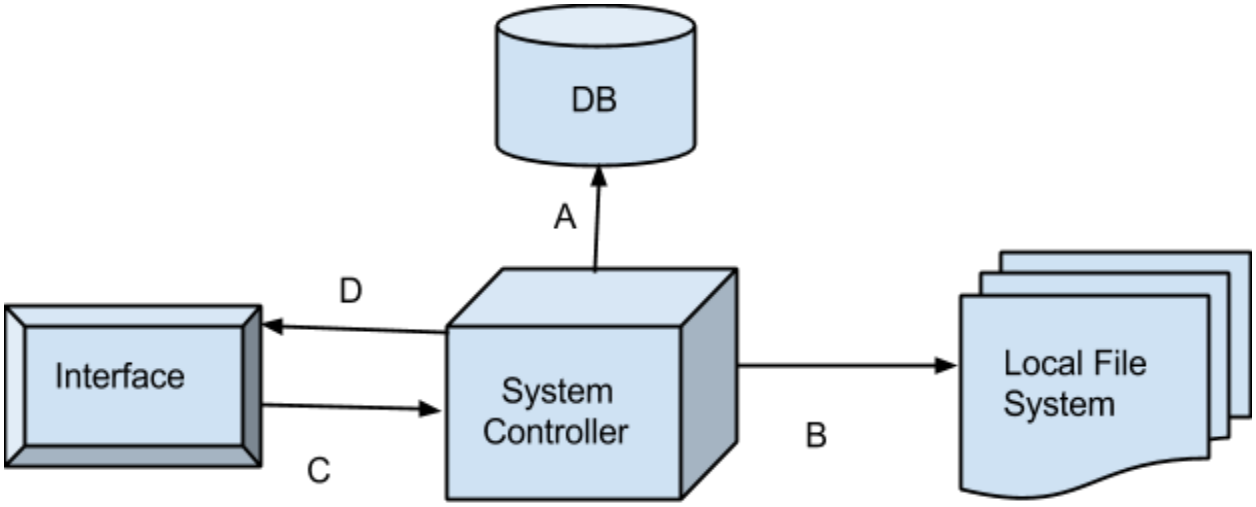
Overview

Lessons Learned will be geared towards creating maximum usefulness from previously created scripts and other source code, both those authored by myself, and any that are pulled from internet resources. Code will be stored in the database and associated with useful search terms for quick reference. Core search terms will include: source of the script, language and version, principles employed within the script (e.g. search algorithms used), date created, last modified, and a description of its function. Also associated with each script will be an (optional) lessons learned text file that outlines the obstacles encountered while writing the piece of code, how long it took and possibly some sample output. This additional information will provide useful reference for the inevitable, "I know I solved this exact problem a few years ago but I can't recall exactly how," situation. A searchable database of past lessons learned, and coding problems overcome, will greatly reduce the time to solve similar problems in the future, as well as provide easy access and reference to scripts and code samples.

Functionality

The goal of the driver application will be to abstract the CRUD access to the data into simple, automated functions. Each script will be tagged with comments that can be extracted to be used as values for the associated database tables. Once in place the system will have a refresh option to scan the provided root directory for any files that have been touched, this will allow the system to be easily updated as new code is added or updated within the file structure. A basic interface will provide access to the database, providing options such as refresh, search, and jump to file (in OS directory where it is stored). Search functions will be highly customizable to refine searches down to specific attributes for fast retrieval of relevant code. I imagine the interface will be a simple command line interface at first but may expand to have a GUI option as well.

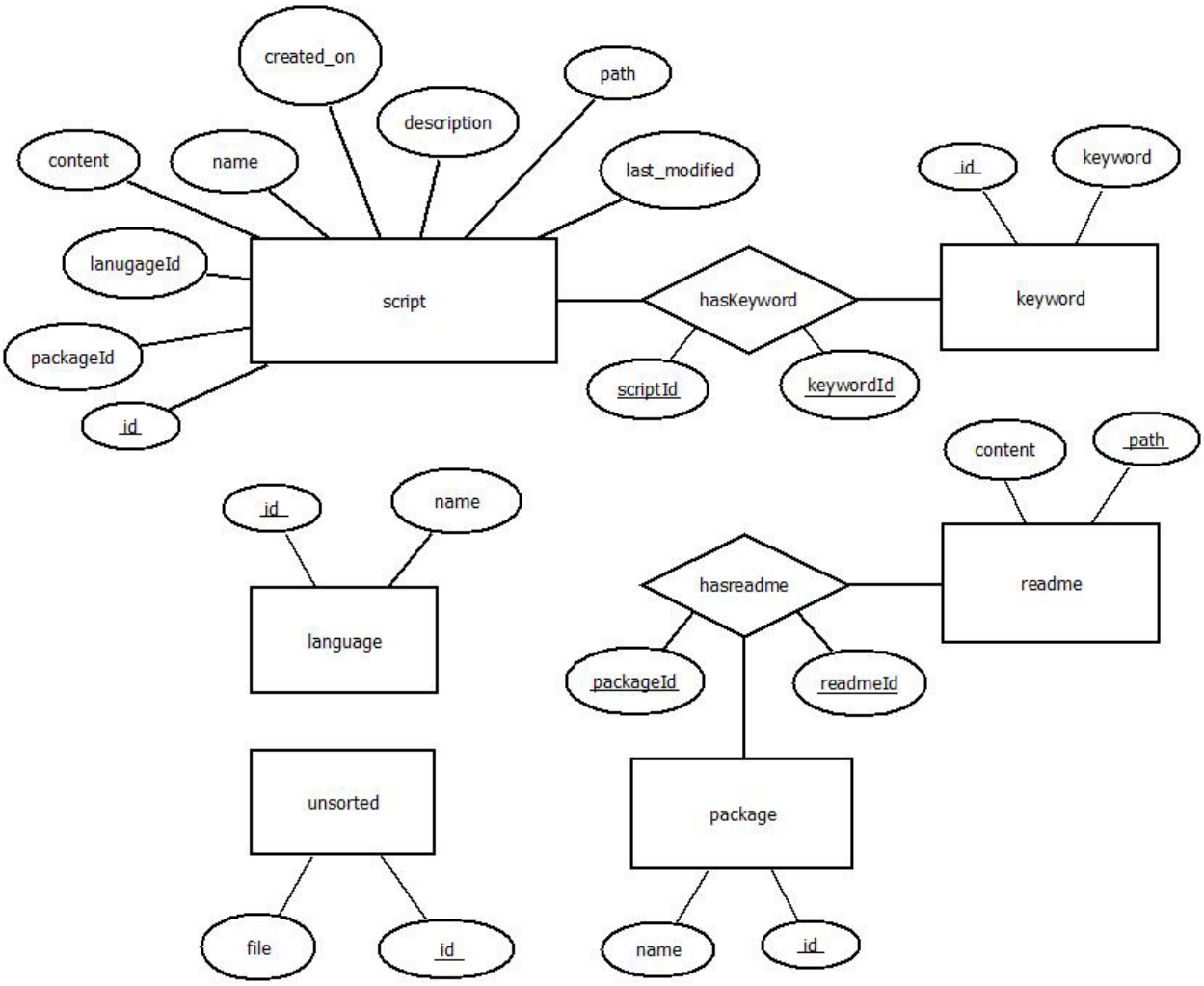
Architecture Overview



A. Data pulled from file system (B) is processed and stored to the database
B. Tags from source code comments and meta data are pulled from the file system for processing and storage
C. Input provided to the controller to perform user interaction
D. Output from the system to the user interface

The system will use an MVC pattern to control the flow of data between the user and system. A local file structure containing the source code will be processed by the controller and stored into the database. Using meta data and comment tags within the source code will remove the necessity for rigid organization within the file structure and place the burden on the controller and database. The user interface will be abstracted from the system to provide alternative display methods and allow for extensibility into GUI or web applications in the future.

ER Diagram of the database



Implementation Status 10/17/2014

Current implementation progress has been focused on developing the groundwork and learning the fundamentals needed to complete the project. Initial setup of the PostgreSQL database has been completed and the implementation language has been chosen to be Python 2.7. After looking at the various options available it seemed that Python offered the rapid deployment option that I was looking for in a language that I am familiar with.

After some initial experimentation I have been able to successfully begin populating my database with a set of prepared scripts. Tags were placed on each test script and a Python script searches through the given file system and finds files that match the set of supported types. The text of the files is then parsed and the tags placed in each script are used to populate the database tables. This is still very much in prototype stage and will require a great deal of tagging on my current collection of scripts as well as expansion of the database builder tool.

One aspect that is going particularly well is the design choice for the project and implementation language. Each test of the system starts with a fresh database and uses the file system information to create and populate the database automatically. Expansion of the features and database tables becomes very simple since there is no manual database creation, entry, or deletion required.

I have not found any reason to alter my E-R diagram yet, though I may need to eliminate the inclusion of external resources. With the amount of work required for this project it may become too difficult to properly manage additional data which may not provide much added value. One of the goals of the system is to have the ability to eventually store entire projects and have their file structure stored. This would allow for easy re-creation of projects if they were lost and provide some interesting analytical options. However, for the purposes of this project it may be wise to put this feature off until a later date because it may introduce too much complexity.

Implementation Status 11/14/2014

Database implementation has progressed beyond the base script table with limited fields to include the ‘package’, ‘keyword’, and ‘haskeyword’ tables. Scripts are scanned in, the keyword, package and description attributes are extracted from the comments in the script files. Relations between keywords and scripts are automatically recorded in the ‘haskeyword’ table.

A basic database query module has been started so that search functionality can be tested as new tables are implemented. Current functions are in place to provide for keyword related queries.

The ER diagram has undergone a major revision to better fit the scope of the project. External reference scripts have been removed and the database will only be populated with my own personal work. Lessons learned documents will also not be stored and have been replaced with README file storage which will serve a similar purpose.

Upcoming work will focus on extracting metadata from files, and implementing the ‘readme’, ‘hasreadme’ and ‘language’ tables to finish off the implementation of the database population. Queries to extract useful reference and statistics will also be added in the final release.

See the README document contained in the source code folder for information on running the current version of the application.

A sample of the database is located in the source folder in the form of a PostgreSQL backup file under the name ‘Phase_2_sample.backup’.

Expected Delivery as of 11/24/2014

At a minimum the final phase of the project will include a database of all (or at least most) of my current software files to date. There will be a tool for refreshing the database which will essentially synchronize my portfolio into the database by looking for recent changes. There will also be a tool for querying the database through a user friendly search interface and provide the results. If time permits the interface to the system will be changed from a command line to a graphical interface.

Final Delivery 12/08/2014

I was able to complete the essential features of the application that were outlined in the Expected Delivery section above. This final delivery contains three primary modules:

1. A database rebuild tool drops the tables, creates, and populates them all to facilitate changes to the database structure for future expansion.

2. An update tool that scans for changes to the files currently tracked in the database and updates the ‘content’ and ‘last_modified’ predicates of the ‘script’ relation.

3. A basic search interface that features a command line tool for simple searches that returns the path and filename of files that match the search. There is also a pure SQL query tool for performing more advanced queries. This module can also generate a basic set of statistics to highlight the quantity of code written in each language.

Lessons Learned employs the following database techniques:

1. Automated population of the database through a file system scan and XML-like tag system for identifying specific information about the files being stored.

2. Foreign key, unique, and null constraints to improve the behaviour and integrity of the database.

3. An update tool that scans the file system and updates the content of changed files and removes rows that are no longer reflected in the file system. This feature will soon be expanded to insert new scripts that are found as well.

4. Multiple roles to restrict access to users performing direct SQL commands to prevent unwanted alteration of the database contents by unauthorized users.

5. Full CRUD access is available through different means, although direct SQL statements are currently restricted in the search interface to SELECT statements only.

Some techniques not employed and rationale:

1. Indices were not used due to the small number or rows in the database. Research on the topic implied that with only a couple hundred rows the database optimization happening under the hood would prevent the index from being employed. As this project grows I will add a second set of relations to house large quantities of reference code, in this case there may be thousands of rows and I will use indices at that time.

2. Triggers and stored procedures were also not used because I was unable to find a practical application for them. I considered using a trigger to store deleted or updated rows as they were affected by the update tool. This could allow for compensating transactions to be run if a mistake is made or I want to revert the contents to a previous state. I decided against this because in reality it will not be practical. My code will also be backed up on GitHub and I do not want to try to replicate the functionality of the repository tool. In the future I may find a useful application for triggers or stored procedures as the database grows, but for now I have found no practical use for them.

3. The original project document specified there should be 100’s to 1000’s of rows in some relations, however this project as it stands has a maximum of a couple hundred within a fairly trivial table. This is largely due to my current body of work only consisting of a couple hundred files and the tedious task required to add the necessary tags into the files. When I apply the database population target to the root directory of all my unsorted software I produce a couple thousand rows in the ‘script’ relation. However much of this code is not mine and produced by code generation wizards or is written by a different author. I wanted this application to specifically reflect my own work and

as such it will take much more time to sort through all of the files and extract only those authored by myself.

Reflection

Overall this was a very enjoyable project to work on. I liked that the assignment was given out early with regular deliverables to set the pace of development. One of my favorite development techniques is to take long breaks to allow new ideas to surface as my subconscious works on the problem. I consistently found that after coming back to the project after a week or two I had fresh ideas and found flaws in the system that I hadn't noticed in the previous cycle. Never before have I worked on such a large scale project with seemingly zero headaches or major roadblocks to slow down progress.

My initial goal was to produce a practical project that could be used for my own continuing education and I feel that in this respect the project was a total success. I look forward to expanding the features of the project by adding in a GUI interface to improve interaction with the database as well as use it as a source for a web application for putting my work on display for potential employers. The statistics generation tool will also be expanded to provide as much useful information as I can find, as well as the aforementioned addition of resource code pulled from the internet. Ultimately I see this project as the first step in accumulating my life's work into a manageable and useful application which can be expanded throughout my career and I am excited to continue working with it to see where I can take it.